

Hypergraph Partitioning for Automatic Memory Hierarchy Management ^a

Sriram Krishnamoorthy

Sriram Krishnamoorthy¹, Umit Catalyurek², Jarek Nieplocha³, Atanas Rountev¹ and P. Sadayappan¹

¹Dept. of CSE, ² Dept. of ECE

The Ohio State University

³ Pacific Northwest National Lab

^aSupported in part by Dept. of Energy & National Science Foundation

Outline

- Background
- Motivation
- Programming Abstractions
- Disk I/O Minimization Problem
- Modeling of the Minimization Problem
- Results
- Related Work
- Conclusion

Background

- Computations with data too large to fit into physical memory
- Out-of-core programming challenging
 - ◆ Schedule computation
 - ◆ Schedule of data movement between disk and main memory
 - ◆ Ensure data required by computation is available in memory
 - ◆ Ensure size of data in memory at any time does not exceed the memory available
- Automatic Memory Hierarchy Management
 - ◆ Relieve burden of out-of-core programming
 - ◆ User specifies computation
 - ◆ Runtime system schedules disk I/O and computation

Tensor Contraction Engine

- Automatic transformation from high-level specification
 - ◆ Chemist specifies computation in high-level mathematical form
 - ◆ Synthesis system transforms it to efficient parallel program
 - ◆ Code is tailored to target machine
 - ◆ Code can be optimized for specific molecules being modeled
- Multi-institutional collaboration (OSU, LSU, U. Waterloo, ORNL, PNNL, U. Florida)
- Significant interest in quantum chemistry community
- Built on top of the Global Arrays library
- Tensor Contractions: Multi-dimensional summations of products of block-sparse arrays

$$p3, p4, p5, p7 : V(100, 100, 60, 60) \quad h1, h2, h6, h8 : O(40, 40, 20, 20)$$

$$i1[h6, p3, h1, p5] += v1[h6, p3, h1, p5]$$

$$i1[h6, p3, h1, p5] += t[p3, p7, h1, h8] * v2[h6, h8, p5, p7]$$

$$i0[p3, p4, h1, h2] += t[p3, p5, h1, h6] * i1[h6, p4, h2, p5]$$

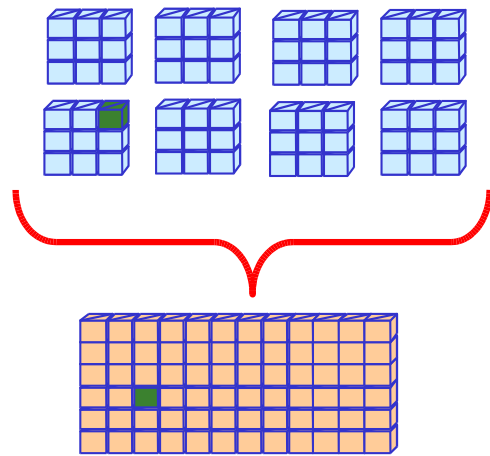
November 15, 2006

SC 2006

Global Arrays Library

Distributed dense arrays that can be accessed through a shared view

Physically Distributed data

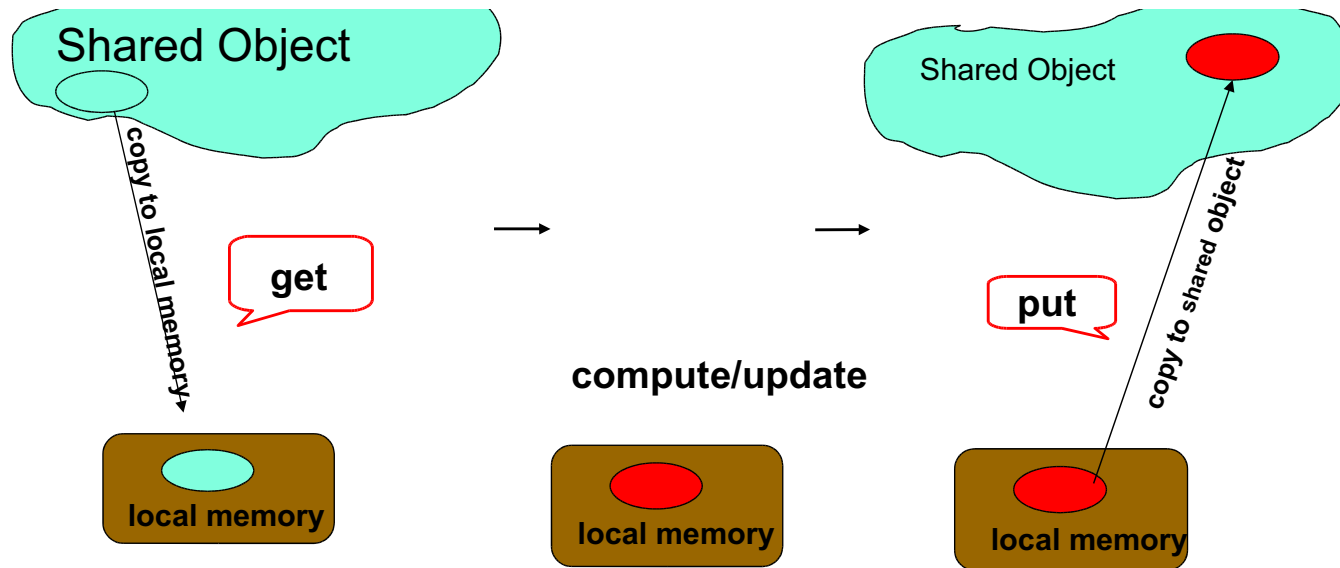


Global Address Space

single shared data structure with indexing.

e.g., access $A(4,3)$ rather than $A_{local}(1,3)$ on process 4

Global Arrays Model of Computation



- Shared memory view for distributed dense arrays
- MPI-Compatible; Currently usable with Fortran, C, C++, Python
- Data locality and granularity control similar to message passing model
- Used in large scale efforts, e.g. NWChem (million+ lines/code)

Pros and Cons of the GA model

- Advantages

- ◆ Provides convenient global-shared view
- ◆ Get-compute-put model ensures user focus on data-locality optimization => good performance
- ◆ Inter-operates with MPI to enable general applicability

- Limitations

- ◆ Only supports dense multi-dimensional arrays
- ◆ Data view more convenient than MPI, but computation specification is still process-centric
- ◆ No support for load balancing of irregular computations

Programming Abstractions

- Decoupled task and data abstractions
- Layered, multi-level data abstraction
 - ◆ Global-shared view and transparent access
 - ◆ Chunked access for efficiency
 - ◆ Multi-dimensional block-sparse array represented as collection of small dense multi-dimensional bricks (chunks)
 - ◆ Bricks distributed among local disks of processors
- Computation Abstraction
 - ◆ Non-process-specific collection of independent tasks
 - ◆ Only non-local data task can access: bricks in global data
 - ◆ Specification of tasks includes the global data accessed : runtime information
 - ◆ Automatic scheduling of computation, communication, and disk I/O

Disk I/O Minimization Problem

- Block-sparse tensors
 - ◆ Irregular data access pattern
 - ◆ Techniques such as tiling — non-trivial to apply
- Data too large to fit into collective physical memory in a parallel system
- Problem characteristics known only at runtime
- Schedule computation and data movement
 - ◆ Minimize disk I/O
 - ◆ Maximize reuse

Hypergraph Partitioning Problem

- Set of vertices/cells and hyperedges/nets with weights
- **Given:** #parts p
- **Solution:** A vertex partition of the hypergraph
- **Objective:** Minimize cost of *cut-net* weights
 - ◆ Weight of cut-nets counted once
 - ◆ Weight of cut-nets counted for every cut
- **Constraint:** Balance for all parts
 - ◆ Sum of vertex weights in each part
 - ◆ Sum of net weights in each part
- Efficient solutions exist (PaToH)
- Mismatch
 - ◆ Typically used in context of parallelization
 - ◆ #parts unknown
 - ◆ No memory limit constraint, only balancing constraint

Modeling Disk I/O Minimization

- **Vertices**: One per task, weight = computation cost
- **Hyper-edges**: One per data-brick, Weight = size of brick (I/O cost)
- Each hyper-edge connects tasks accessing the brick
- **Minimize**: Sum of cut-net weights, counting every cut
- **Constraint**: Balance net weights
 - ◆ Balancing computation cost (vertex weights) unnecessary
 - ◆ Ensures a feasible solution, if one exists
- #parts - Number of stages in the computation
- Dynamically determine #parts
 - ◆ Modify recursive procedure of hypergraph partitioning
 - ◆ Stop further partitioning when memory constraint satisfied

Read-Once Partitioning

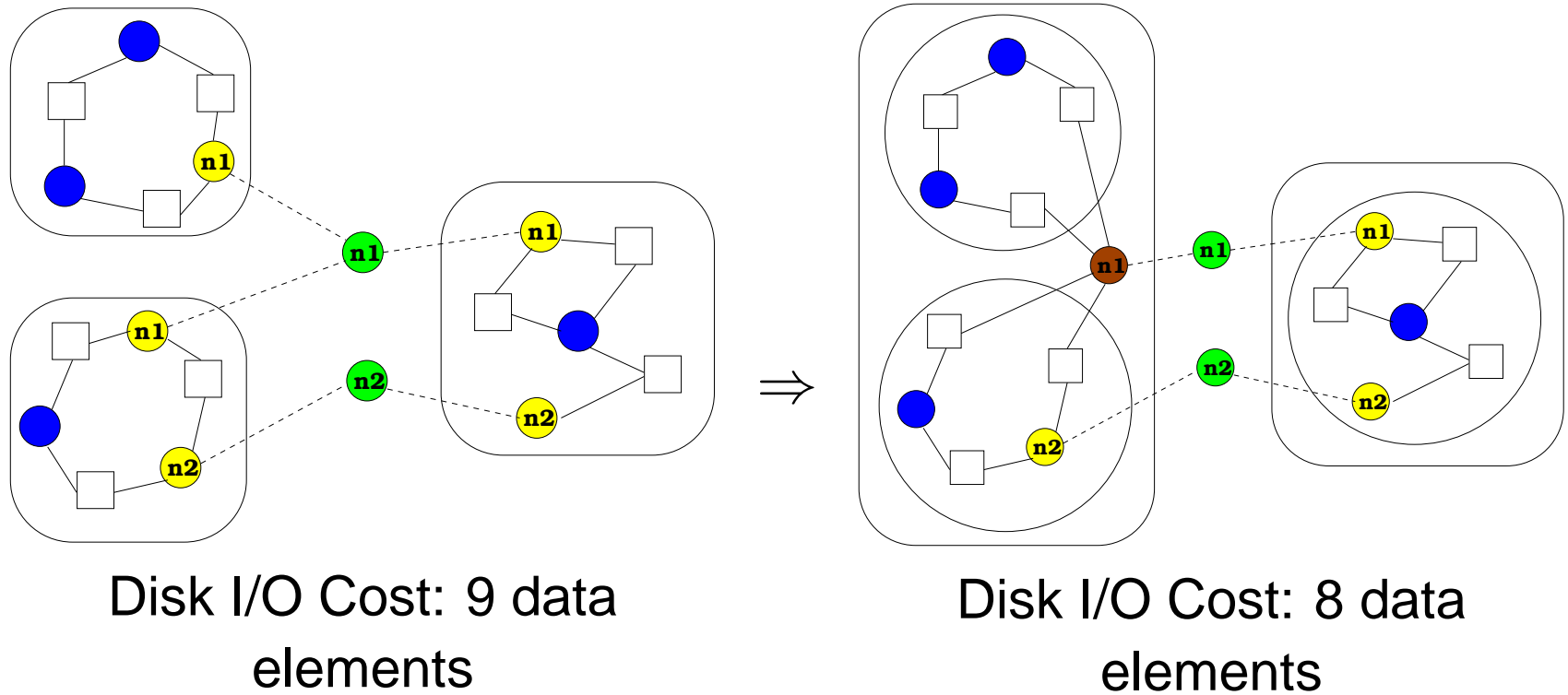
- One-level Partitioning

- ◆ Memory usage = Sum of data sizes accessed in a part
- ◆ No refined reuse relationships
- ◆ All tasks within a part have reuse, and none outside

- Read-Once Partitioning

- ◆ Group tasks into steps
- ◆ Identify data common across steps and load into memory
- ◆ For each step, read non-common (step-exclusive) data, process tasks, and write/discard step-exclusive data
- ◆ Better utilization of memory available -> reduced disk I/O

Read-once Partitioning Example

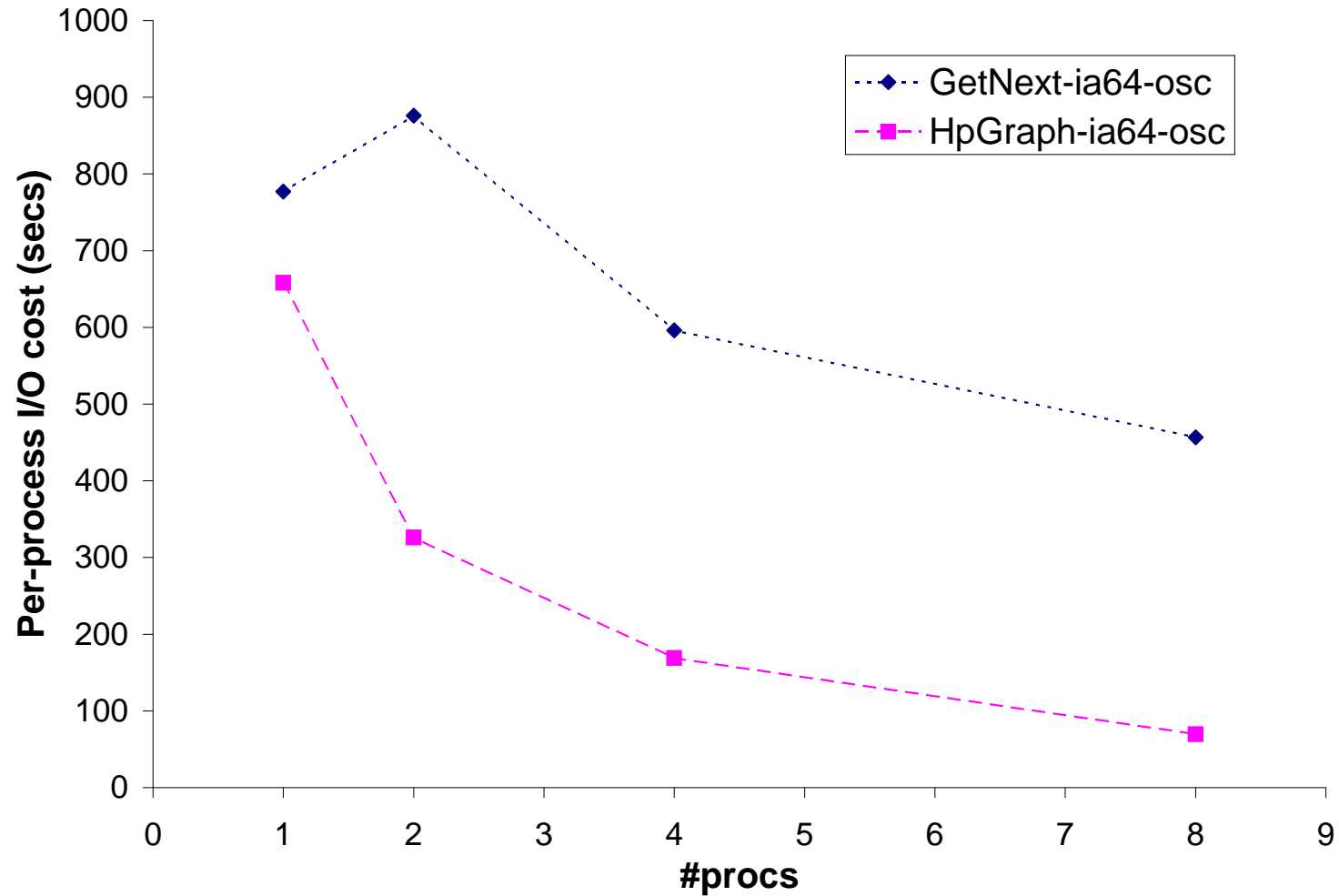


Results - Configuration

| | ia64-osc | p4-osc | ia64-pnl |
|-----------------|----------------|----------------|----------------|
| Processor | Dual Itanium-2 | Dual Pentium 4 | Dual Itanium-2 |
| Clock Frequency | 900MHz | 2.4GHz | 1GHz |
| Physical Memory | 4GB | 4GB | 6GB |
| Local Disk | 80GB | 80GB | 80GB |
| Messaging Layer | GM | VAPI | GM |

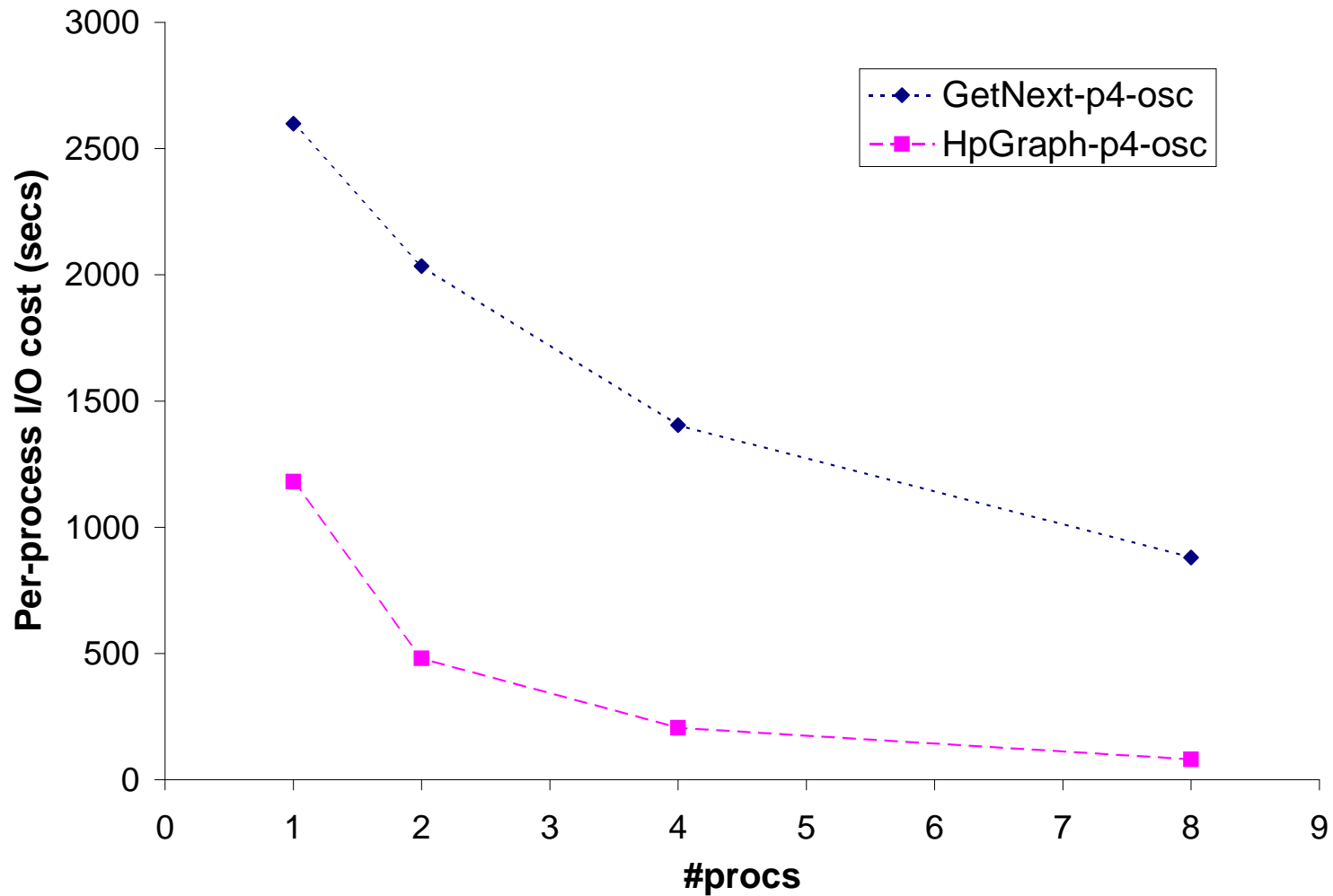
- One process per node used in all experiments
- Alternative Scheme: GetNext
 - ◆ Replicate, compute, reconcile model
 - ◆ Based on codes in NWChem
- Evaluation on CCD sub-computation described earlier

Results - Disk I/O Cost on ia64-osc



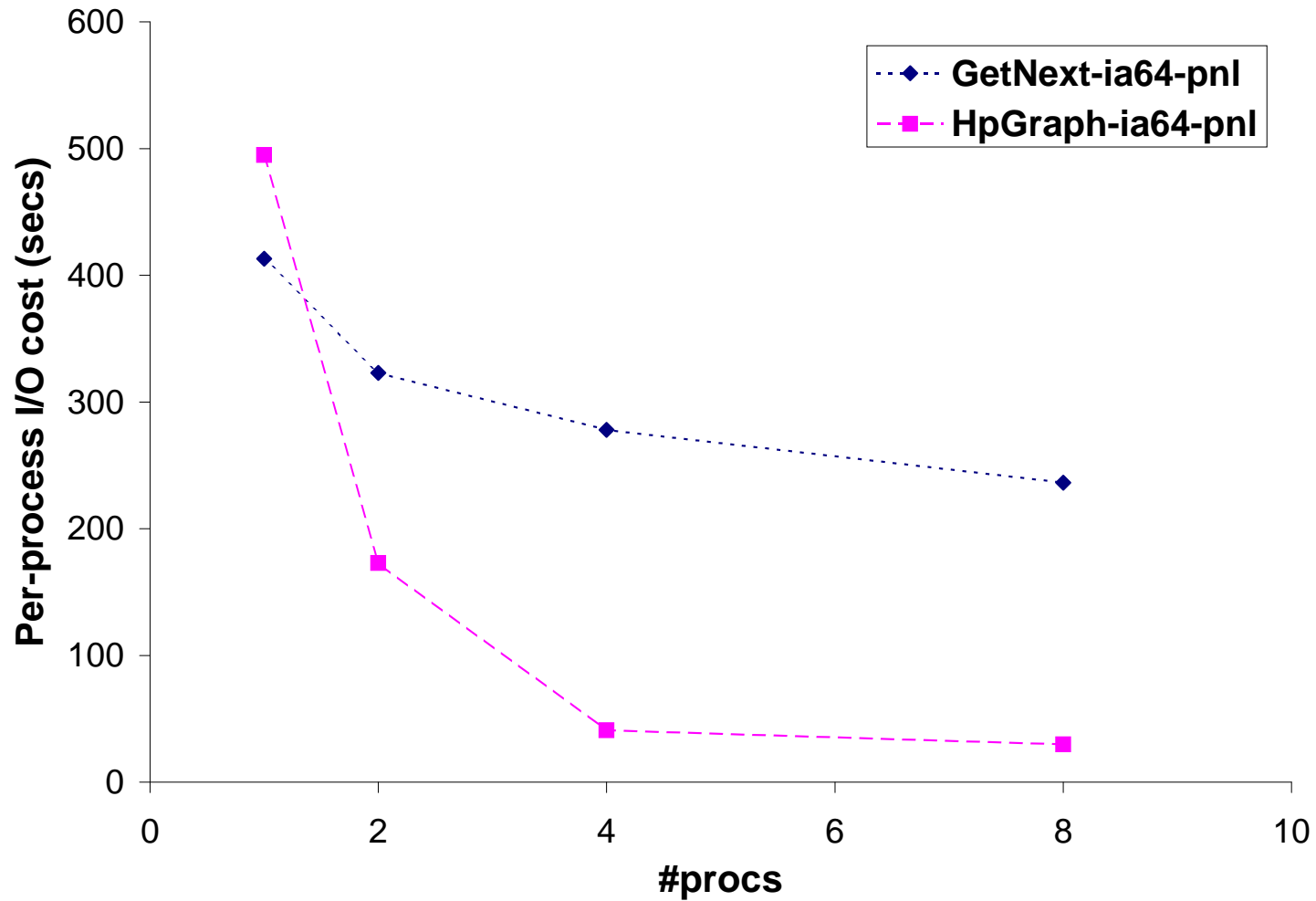
Factor of Improvement: 6.5

Results - Disk I/O Cost on p4-osc



Factor of Improvement: 11.0

Results - Disk I/O Cost on ia64-pnl



Factor of Improvement: 7.9

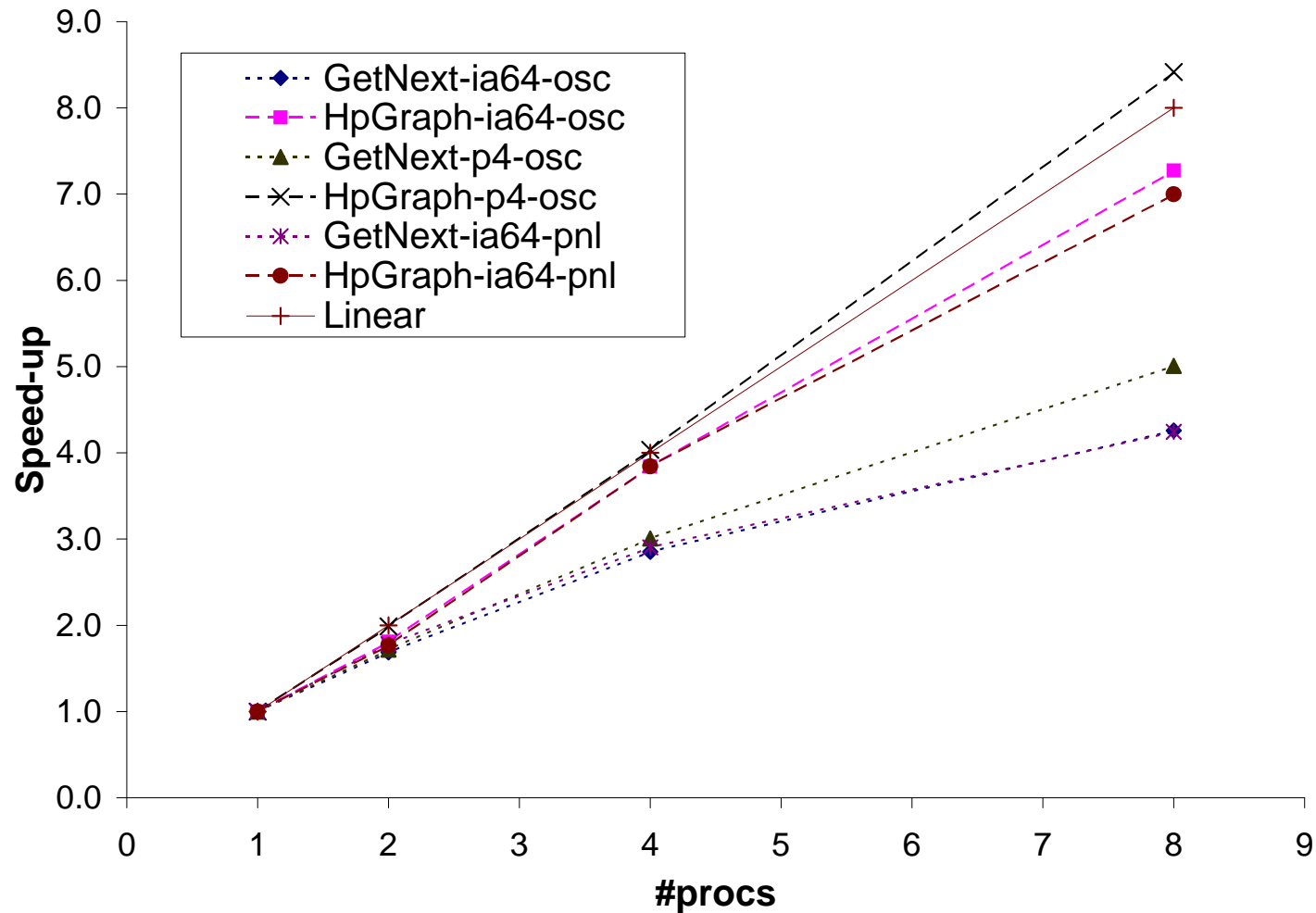
Results - Total Execution Time

| System | Scheme | nprocs | | | |
|----------|---------|--------|------|------|------|
| | | 1 | 2 | 4 | 8 |
| ia64-osc | GetNext | 9710 | 5760 | 3403 | 2281 |
| | HpGraph | 9244 | 5110 | 2408 | 1271 |
| p4-osc | GetNext | 13717 | 7988 | 4562 | 2739 |
| | HpGraph | 11700 | 5886 | 2899 | 1390 |
| ia64-pnl | GetNext | 7928 | 4453 | 2731 | 1868 |
| | HpGraph | 7564 | 4283 | 1968 | 1081 |

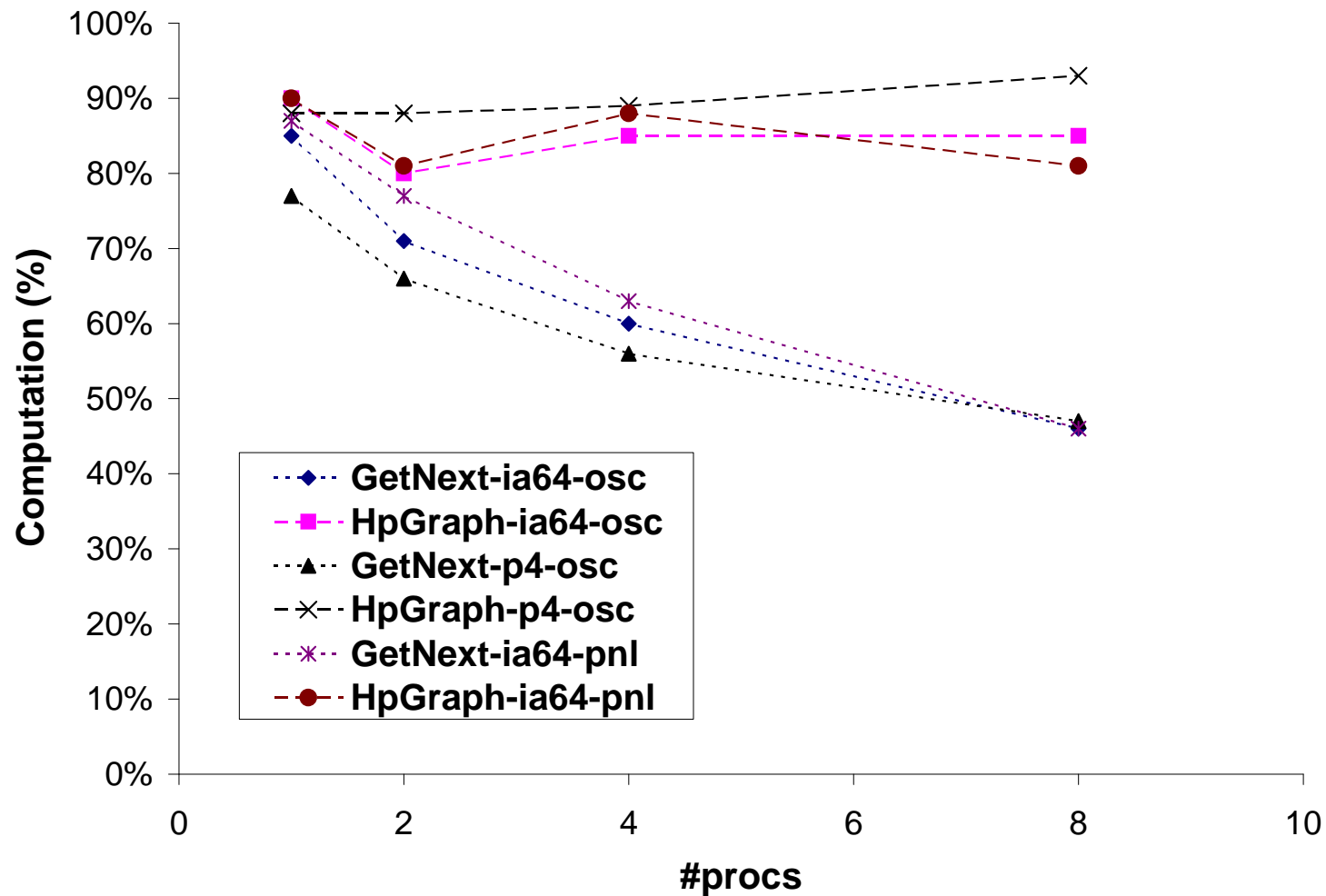
Table 1: Turnaround times for the CCD sub-computation

Sequential Speedup on p4-osc : 14.7%

Results - Execution Time Speed-ups



Results - Percentage of Computation overhead



Related Work

- Sparse or block-sparse matrices in parallel libraries supporting sparse linear algebra
 - ◆ Aztec, PETSc etc.
 - ◆ Load balancing
- Dense tiling - challenging in this context
- Application of hypergraph partitioning
 - ◆ Primarily used for parallelization
- Unaware of runtime support for:
 - ◆ Flexible global-shared abstractions for semi-structured data
 - ◆ Locality-aware scheduling of parallel tasks
 - ◆ Transparent memory hierarchy management

Conclusion

- Abstractions for locality-aware transparent memory hierarchy management
 - ◆ Data chunked for efficient access
 - ◆ Computation specification includes locality
- Modeled disk I/O minimization as hypergraph partitioning
- Partitioning to reduce disk I/O while attaining a feasible solution
- Demonstrated sequential speedup
- Demonstrated better scalability

Questions?
